Assignment 1

Sir

Submitted By: Vikhyat Bansal [CB.EN.U4AIE21076]



Q1. Write a note on BST (Binary Search Tree) give an example for inorder traversal and write code in java/python for inorder traversal.

Ans. BST (Binary Search Tree) is a data structure and a type of binary tree where the value in node on the left of the parent node is always less and in similar fashion the value in node on the right of the parent node is always greater.



Inorder traversal follows: Left, Root, Right

Inorder traversal of the above tree:   
  
25,30,35,40,45,50,60

Code:

// *binary search tree*

import *java*.*util*.*\**;

*public* *class* BST {

*static* *class* Node {

        int data;

        Node left, right;

*public* *Node*(int data) {

            this.*data* = data;

            left = right = null;

        }

    }

*static* Node root;

*static* Node node = root;

*static* void *insert*(int data) {

        root = *insertRec*(root, data);

    }

*static* Node *insertRec*(Node root, int data) {

*if* (root == null) {

            root = *new* *Node*(data);

*return* root;

        }

*if* (data < root.*data*) {

            root.*left* = *insertRec*(root.*left*, data);

        } *else* *if* (data > root.*data*) {

            root.*right* = *insertRec*(root.*right*, data);

        }

*return* root;

    }

*static* void *inorder*() {

*inorderRec*(root);

    }

*static* void *inorderRec*(Node root) {

*if* (root != null) {

*inorderRec*(root.*left*);

            System.*out*.*println*(root.*data*);

*inorderRec*(root.*right*);

        }

    }

*public* *static* void *main*(String[] args) {

        BST tree = *new* *BST*();

        tree.*insert*(150);

        tree.*insert*(100);

        tree.*insert*(200);

        tree.*insert*(75);

        tree.*insert*(125);

        tree.*insert*(175);

        tree.*insert*(250);

        tree.*insert*(60);

        tree.*insert*(90);

        tree.*insert*(110);

        tree.*insert*(130);

        tree.*insert*(170);

        tree.*insert*(190);

        tree.*insert*(225);

        tree.*insert*(300);

        tree.*insert*(30);

        tree.*insert*(70);

        tree.*insert*(80);

        tree.*insert*(95);

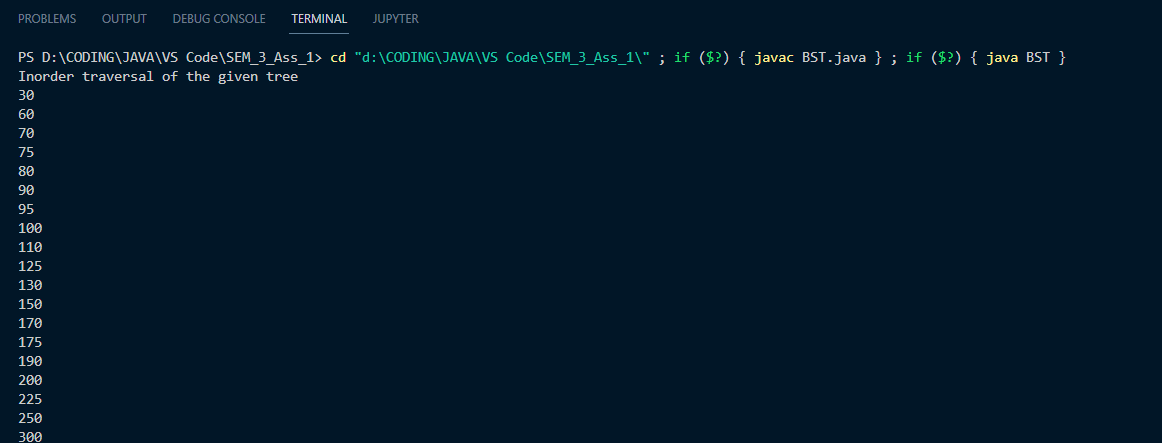
        System.*out*.*println*("Inorder traversal of the given tree");

        tree.*inorder*();

    }

}

Output:



Q2. Write a code to check whether the given binary tree is a BST?

a) root = 10, root.left = 4, root.right = 11, root.left.left = 1, root.left.right = 2

b) root = 5, root.left = 3, root.right, root.left.left = 2, root.left.right = 4

Ans. CODE:

*class* BTreeLL{

// *A binary tree node has data, pointer to*

// *left child and a pointer to right child*

*static* *class* Node

{

    int data;

    Node left, right;

    // *Constructor*

*Node*(int data)

    {

        this.*data* = data;

        left = null;

        right = null;

    }

}

*static* Node root;

*static* Node node = root;

// *Inorder traversal of a binary tree*

*static* void *inorderTraversal*(Node node)

    {

*if* (node != null)

        {

*inorderTraversal*(node.*left*);

            System.*out*.*print*(node.*data* + " ");

*inorderTraversal*(node.*right*);

        }

    }

//*check whether the tree is binary search or not*

*static* boolean *isBST*(Node root) {

*if* (root == null) {

*return* true;

    }

*if* (root.*left* != null && root.*left*.*data* > root.*data*) {

*return* false;

    }

*if* (root.*right* != null && root.*right*.*data* < root.*data*) {

*return* false;

    }

*if* (!*isBST*(root.*left*) || !*isBST*(root.*right*)) {

*return* false;

    }

*return* true;

}

// *Driver code*

*public* *static* void *main*(String args[])

{

    root = *new* *Node*(10);

    root.*left* = *new* *Node*(4);

    root.*left*.*left* = *new* *Node*(1);

    root.*left*.*right* = *new* *Node*(2);

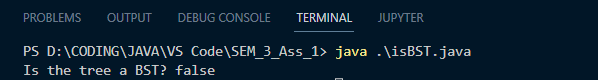
    root.*right* = *new* *Node*(11);

    System.*out*.*println*("Is the tree a BST? " + *isBST*(root));

}

}

Output:



Q2. B)

CODE:

*class* BTreeLL{

// *A binary tree node has data, pointer to*

// *left child and a pointer to right child*

*static* *class* Node

{

    int data;

    Node left, right;

    // *Constructor*

*Node*(int data)

    {

        this.*data* = data;

        left = null;

        right = null;

    }

}

*static* Node root;

*static* Node node = root;

// *Inorder traversal of a binary tree*

*static* void *inorderTraversal*(Node node)

    {

*if* (node != null)

        {

*inorderTraversal*(node.*left*);

            System.*out*.*print*(node.*data* + " ");

*inorderTraversal*(node.*right*);

        }

    }

//*check whether the tree is binary search or not*

*static* boolean *isBST*(Node root) {

*if* (root == null) {

*return* true;

    }

*if* (root.*left* != null && root.*left*.*data* > root.*data*) {

*return* false;

    }

*if* (root.*right* != null && root.*right*.*data* < root.*data*) {

*return* false;

    }

*if* (!*isBST*(root.*left*) || !*isBST*(root.*right*)) {

*return* false;

    }

*return* true;

}

// *Driver code*

*public* *static* void *main*(String args[])

{

    root = *new* *Node*(5);

    root.*left* = *new* *Node*(3);

    root.*left*.*left* = *new* *Node*(2);

    root.*left*.*right* = *new* *Node*(4);

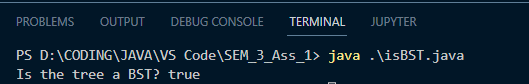
    root.*right* = *new* *Node*(7);

    System.*out*.*println*("Is the tree a BST? " + *isBST*(root));

}

}

Output:



Q3. Create a BST with level 5

a) display the values of the nodes on the left side of the tree.

b) display the values of the nodes on the right side of the tree.

CODE:

// *Java program to delete element*

// *in binary tree*

import *java*.*util*.*LinkedList*;

import *java*.*util*.*Queue*;

*class* Left{

// *A binary tree node has data, pointer to*

// *left child and a pointer to right child*

*static* *class* Node

{

    int data;

    Node left, right;

    // *Constructor*

*Node*(int data)

    {

        this.*data* = data;

        left = null;

        right = null;

    }

}

*static* Node root;

*static* Node node = root;

// *Inorder traversal of a binary tree*

*static* void *inorderTraversal*(Node node)

    {

*if* (node != null)

        {

*inorderTraversal*(node.*left*);

            System.*out*.*print*(node.*data* + " ");

*inorderTraversal*(node.*right*);

        }

    }

*static* boolean *isBST*(Node root) {

*if* (root == null) {

*return* true;

    }

*if* (root.*left* != null && root.*left*.*data* > root.*data*) {

*return* false;

    }

*if* (root.*right* != null && root.*right*.*data* < root.*data*) {

*return* false;

    }

*if* (!*isBST*(root.*left*) || !*isBST*(root.*right*)) {

*return* false;

    }

*return* true;

}

*static* int upper\_level = 0;

*static* void *leftSide*(Node node, int level)

{

    // *Base Case*

*if* (node == null)

*return*;

    // *If this is the first node of its level*

*if* (upper\_level < level) {

        System.*out*.*print*(node.*data* + " ");

        upper\_level = level;

    }

    // *Recur for left and right subtrees*

*leftSide*(node.*left*, level + 1);

*leftSide*(node.*right*, level + 1);

}

// *A wrapper over leftViewUtil()*

*static* void *leftView*()

{

    upper\_level = 0;

*leftSide*(root, 1);

}

*public* *static* void *main*(String args[])

{

    root = *new* *Node*(100);

    root.*left* = *new* *Node*(50);

    root.*left*.*left* = *new* *Node*(30);

    root.*left*.*right* = *new* *Node*(70);

    root.*right* = *new* *Node*(150);

    root.*right*.*left* = *new* *Node*(140);

    root.*right*.*right* = *new* *Node*(180);

    root.*right*.*right*.*left* = *new* *Node*(160);

    root.*left*.*left*.*left* = *new* *Node*(20);

    root.*left*.*left*.*right* = *new* *Node*(40);

    root.*right*.*right*.*left*.*right* = *new* *Node*(170);

    root.*right*.*right*.*left*.*left* = *new* *Node*(155);

    root.*right*.*right*.*right* = *new* *Node*(200);

    root.*right*.*right*.*left*.*right*.*left* = *new* *Node*(165);

    root.*right*.*right*.*left*.*right*.*right* = *new* *Node*(175);

    root.*left*.*left*.*right*.*left* = *new* *Node*(35);

    root.*left*.*left*.*right*.*right* = *new* *Node*(45);

    root.*left*.*left*.*right*.*left*.*right* = *new* *Node*(37);

    root.*left*.*left*.*right*.*left*.*left* = *new* *Node*(33);

    System.*out*.*println*("Is the tree a BST? " + *isBST*(root));

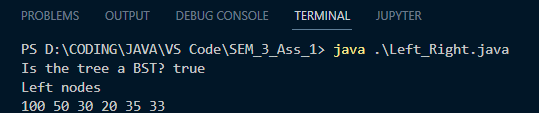
    System.*out*.*println*("Left nodes");

*leftView*();

}

}

Output:



b) CODE:

// *Java program to delete element*

// *in binary tree*

import *java*.*util*.*LinkedList*;

import *java*.*util*.*Queue*;

*class* Left{

// *A binary tree node has data, pointer to*

// *left child and a pointer to right child*

*static* *class* Node

{

    int data;

    Node left, right;

    // *Constructor*

*Node*(int data)

    {

        this.*data* = data;

        left = null;

        right = null;

    }

}

*static* Node root;

*static* Node node = root;

// *Inorder traversal of a binary tree*

*static* void *inorderTraversal*(Node node)

    {

*if* (node != null)

        {

*inorderTraversal*(node.*left*);

            System.*out*.*print*(node.*data* + " ");

*inorderTraversal*(node.*right*);

        }

    }

*static* boolean *isBST*(Node root) {

*if* (root == null) {

*return* true;

    }

*if* (root.*left* != null && root.*left*.*data* > root.*data*) {

*return* false;

    }

*if* (root.*right* != null && root.*right*.*data* < root.*data*) {

*return* false;

    }

*if* (!*isBST*(root.*left*) || !*isBST*(root.*right*)) {

*return* false;

    }

*return* true;

}

*static* int upper\_level = 0;

*static* void *rightSide*(Node node, int level)

{

    // *Base Case*

*if* (node == null)

*return*;

    // *If this is the first node of its level*

*if* (upper\_level < level) {

        System.*out*.*print*(node.*data* + " ");

        upper\_level = level;

    }

    // *Recur for left and right subtrees*

*rightSide*(node.*right*, level + 1 );

*rightSide*(node.*left*, level + 1);

}

*static* void *rightView*()

{

    upper\_level = 0;

*rightSide*(root, 1);

}

*public* *static* void *main*(String args[])

{

    root = *new* *Node*(100);

    root.*left* = *new* *Node*(50);

    root.*left*.*left* = *new* *Node*(30);

    root.*left*.*right* = *new* *Node*(70);

    root.*right* = *new* *Node*(150);

    root.*right*.*left* = *new* *Node*(140);

    root.*right*.*right* = *new* *Node*(180);

    root.*right*.*right*.*left* = *new* *Node*(160);

    root.*left*.*left*.*left* = *new* *Node*(20);

    root.*left*.*left*.*right* = *new* *Node*(40);

    root.*right*.*right*.*left*.*right* = *new* *Node*(170);

    root.*right*.*right*.*left*.*left* = *new* *Node*(155);

    root.*right*.*right*.*right* = *new* *Node*(200);

    root.*right*.*right*.*left*.*right*.*left* = *new* *Node*(165);

    root.*right*.*right*.*left*.*right*.*right* = *new* *Node*(175);

    root.*left*.*left*.*right*.*left* = *new* *Node*(35);

    root.*left*.*left*.*right*.*right* = *new* *Node*(45);

    root.*left*.*left*.*right*.*left*.*right* = *new* *Node*(37);

    root.*left*.*left*.*right*.*left*.*left* = *new* *Node*(33);

    System.*out*.*println*("Is the tree a BST? " + *isBST*(root));

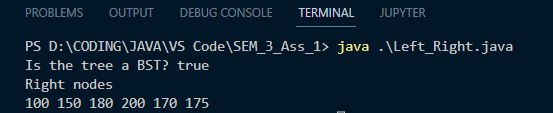
    System.*out*.*println*("Right nodes");

*rightView*();

}

}

Output:



Q4. For the previous binary search tree display all paths from root to leaf nodes.

Code:

*class* path {

*static* *class* Node

{

    int data;

    Node left, right;

*Node*(int data)

    {

        this.*data* = data;

        left = null;

        right = null;

    }

}

*static* Node root;

*static* Node node = root;

*static* void *inorderTraversal*(Node node)

    {

*if* (node != null)

        {

*inorderTraversal*(node.*left*);

            System.*out*.*print*(node.*data* + " ");

*inorderTraversal*(node.*right*);

        }

    }

*static* boolean *isBST*(Node root) {

*if* (root == null) {

*return* true;

    }

*if* (root.*left* != null && root.*left*.*data* > root.*data*) {

*return* false;

    }

*if* (root.*right* != null && root.*right*.*data* < root.*data*) {

*return* false;

    }

*if* (!*isBST*(root.*left*) || !*isBST*(root.*right*)) {

*return* false;

    }

*return* true;

}

*static* void *printPath*(Node node) {

    int path[] = *new* int[1000];

*printPathsRecurs*(node, path, 0);

}

*static* void *printPathsRecurs*(Node node, int path[], int pathLen) {

*if* (node == null)

*return*;

    /\* *append this node to the path array* \*/

    path[pathLen] = node.*data*;

    pathLen++;

    /\* *it's a leaf, so print the path that led to here* \*/

*if* (node.*left* == null && node.*right* == null)

*printArr*(path, pathLen);

*else* {

        /\* *otherwise try both subtrees* \*/

*printPathsRecurs*(node.*left*, path, pathLen);

*printPathsRecurs*(node.*right*, path, pathLen);

    }

}

/\* *Utility that prints out an array on a line.* \*/

*static* void *printArr*(int integ[], int len) {

    int i;

*for* (i = 0; i < len; i++) {

        System.*out*.*print*(integ[i] + " ");

    }

    System.*out*.*println*("");

    System.*out*.*println*();

}

*public* *static* void *main*(String[] args) {

    root = *new* *Node*(100);

    root.*left* = *new* *Node*(50);

    root.*left*.*left* = *new* *Node*(30);

    root.*left*.*right* = *new* *Node*(70);

    root.*right* = *new* *Node*(150);

    root.*right*.*left* = *new* *Node*(140);

    root.*right*.*right* = *new* *Node*(180);

    root.*right*.*right*.*left* = *new* *Node*(160);

    root.*left*.*left*.*left* = *new* *Node*(20);

    root.*left*.*left*.*right* = *new* *Node*(40);

    root.*right*.*right*.*left*.*right* = *new* *Node*(170);

    root.*right*.*right*.*left*.*left* = *new* *Node*(155);

    root.*right*.*right*.*right* = *new* *Node*(200);

    root.*right*.*right*.*left*.*right*.*left* = *new* *Node*(165);

    root.*right*.*right*.*left*.*right*.*right* = *new* *Node*(175);

    root.*left*.*left*.*right*.*left* = *new* *Node*(35);

    root.*left*.*left*.*right*.*right* = *new* *Node*(45);

    root.*left*.*left*.*right*.*left*.*right* = *new* *Node*(37);

    root.*left*.*left*.*right*.*left*.*left* = *new* *Node*(33);

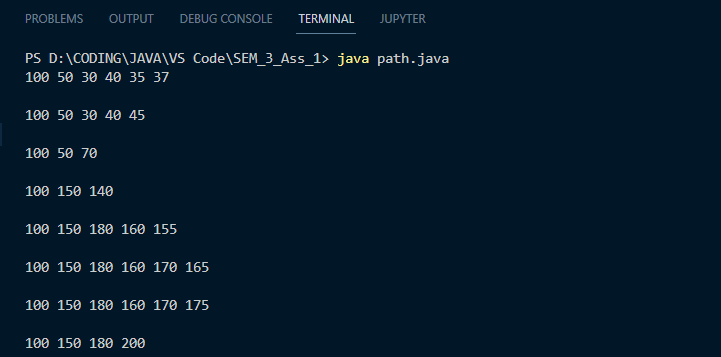
    System.*out*.*println*("All the paths from root to leaf are: ");

*printPath*(root);

}

}

Output:



THANK YOU